Algorithmen und Datenstrukturen 2

Sommersemester 2007
3. Vorlesung

Peter F. Stadler

Universität Leipzig
Institut für Informatik
studla@bioinf.uni-leipzig.de

Algorithmen für Graphen

Fragestellungen:

- Suche von Objekten in Graphen
- Suche von kurzen Wegen in Graphen
- Vollständiges Durchlaufen von Graphen
  - Besuch aller Knoten
  - Besuch aller Kanten
- Färbungsprobleme

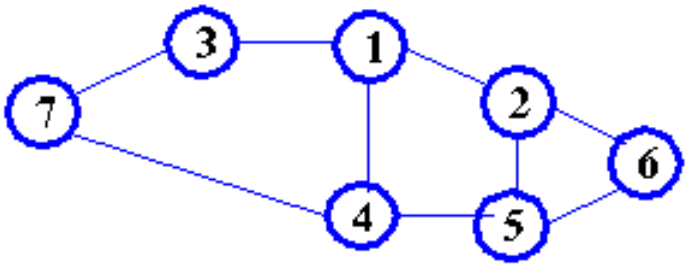
Whdlg: Graphen

Graphen sind zur Repräsentation von Problemen vielseitig verwendbar, z.B.

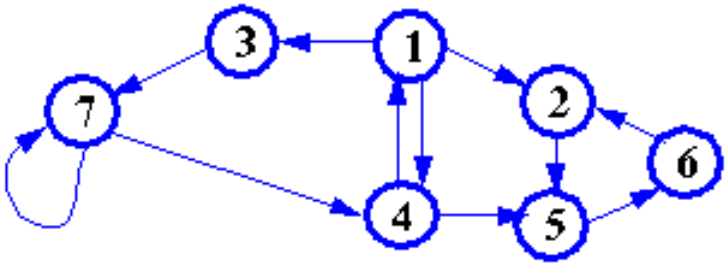
- Städte: Verbindungswege
- Personen: Relationen zwischen ihnen
- Rechner: Verbindungen
- Aktionen: zeitliche Abhängigkeiten

Graph: Menge von Knoten (Vertices) und Kanten (Edges)

- ungerichtete Graphen
- gerichtete Graphen (Digraph, Directed graph)
- gerichtete, azyklische Graphen (DAG, Directed Acyclic Graph)



ungerichteter Graph G_u



gerichteter Graph G_g

Definitionen

G = (V, E) heißt ungerichteter Graph gdw.:

- V ist eine endliche, nichtleere Menge. V heißt Knotenmenge, Elemente von V heißen Knoten
- E ist eine Menge von ein- oder zweielementigen Teilmengen von V. E heißt Kantenmenge, ein Paar $\{u,v\} \in E$ heißt Kante
- Eine Kante {u} heißt Schlinge
- Zwei Knoten u und v heißen benachbart (adjazent) gdw.: $\{u,v\} \in E$ oder $(u=v) \wedge \{u\} \in E$.

Sei G = (V,E) ein ungerichteter Graph. Wenn E keine Schlinge enthält, so heißt G schlingenlos.

Bemerkung: Im weiteren werden wir Kanten {u,v} als Paare (u,v) oder (v,u) und Schlingen {u} als Paar (u,u) schreiben, um spätere gemeinsame Definitionen für ungerichtete und gerichtete Graphen nicht differenzieren und notationell unterscheiden zu müssen.

Seien $G = (V_G, E_G)$ und $H = (V_H, E_H)$ ungerichtete Graphen.

- H heißt Teilgraph von G ($H \subset G$) gdw.: $V_G \supset V_H$ und $E_G \supset E_H$
- H heißt vollständiger Teilgraph von G :

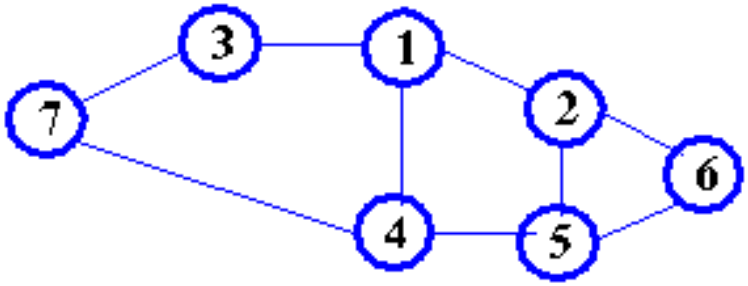
  $H \subset G$ und $[(u,v) \in E_G$ mit $u,v \in V_H \Rightarrow (u,v) \in E_H]$.

Beispiele ungerichteter Graphen

Beispiel 1

- $G = (V_G, E_G)$ mit
- $V_G = \{1, 2, 3, 4\}$,
- $E_G = \{(1, 2), (1, 3), (1, 4), (2, 3), (2, 4), (3, 4)\}$

Beispiel 2



ungerichteter Graph G_u

Definitionen (2)

G = (V,E) heißt gerichteter Graph (Directed Graph, Digraph) gdw.:

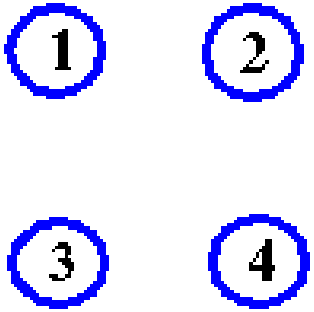
- V ist endliche Menge. V heißt Knotenmenge, Elemente von V heißen Knoten.
- $E \subset V \times V$ heißt Kantenmenge, Elemente von E heißen Kanten.

Schreibweise: (u, v) oder $u \rightarrow v$. Dabei ist u die Quelle, v das Ziel der Kante $u \rightarrow v$.

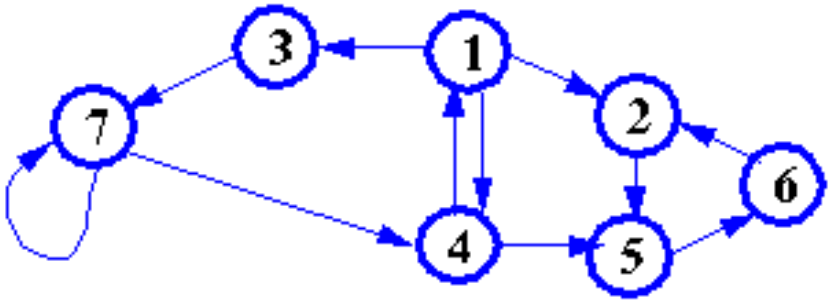
- Eine Kante (u, u) heißt Schlinge.

Beispiel 1

- $G = (V_G, E_G)$ mit $V_G = \{1, 2, 3, 4\}$ und $E_G = \{1 \rightarrow 2,$
  $1 \rightarrow 3, 1 \rightarrow 4, 2 \rightarrow 3, 2 \rightarrow 4, 3 \rightarrow 4\}$



Beispiel 2



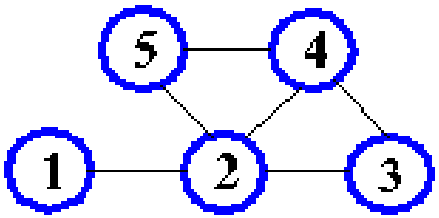
gerichteter Graph G_g

Definitionen (3)

Sei $G = (V,E)$ ein (un-)gerichteter Graph und $k = (v_o, ..., v_n) \in V^{n+1}$

- k heißt *Kantenfolge* der Länge n von v_o nach v_n, wenn für alle $i \in \{0, ..., n-1\}$ gilt: $(v_i, v_{i+1}) \in E$. Im gerichteten Fall ist v_o der Startknoten und v_n der Endknoten, im ungerichteten Fall sind v_o und v_n die Endknoten von k.
- $v_1, ..., v_{n-1}$ sind die *inneren Knoten* von k. Ist $v_o = v_n$, so ist die Kantenfolge *geschlossen*.
- k heißt *Kantenzug* der Länge n von v_o nach v_n, wenn k Kantenfolge der Länge n von v_o nach v_n ist und wenn für alle $i, j \in \{0, ..., n-1\}$ mit $i \neq j$ gilt: $(v_i, v_{i+1}) \neq (v_j, v_{j+1})$.
- k heißt *Weg* (Pfad) der Länge n von v_o nach v_n, wenn k Kantenfolge der Länge n von v_o nach v_n ist und wenn für alle $i, j \in \{0, ..., n\}$ mit $i \neq j$ gilt: $v_i \neq v_j$
- k heißt *Zyklus* oder Kreis der Länge n, wenn k geschlossene Kantenfolge der Länge n von v_o nach v_n und wenn $k' = (v_o, ..., v_{n-1})$ ein Weg ist. Ein Graph ohne Zyklus heißt kreisfrei oder *azyklisch*. Ein gerichteter azyklischer Graph heißt auch *DAG* (*Directed Acyclic Graph*)
- Graph ist *zusammenhängend*, wenn zwischen je 2 Knoten ein Kantenzug existiert

Definitionen (4)



Kantenfolge:
Kantenzug:
Weg:
Zyklus:

Sei G = (V, E) (un)gerichteter Graph, k Kantenfolge von v nach w. Dann gibt es einen Weg von v nach w.

Sei G = (V, E) ein gerichteter Graph

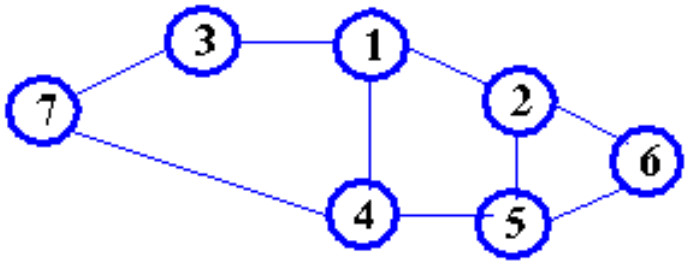
- *Eingangsgrad*: $eg(v) = |\{v' \mid (v', v) \in E\}|$
- *Ausgangsgrad*: $ag(v) = |\{v' \mid (v, v') \in E\}|$
- G heißt *gerichteter Wald*, wenn G zyklenfrei ist und für alle Knoten v gilt $eg(v) \leq 1$. Jeder Knoten v mit $eg(v)=0$ ist eine Wurzel des Waldes.
- *Aufspannender Wald (Spannwald)* von G: gerichteter Wald $W=(V,F)$ mit $F \subseteq E$

Gerichteter Baum (Wurzelbaum): gerichteter Wald mit genau 1 Wurzel

- für jeden Knoten v eines gerichteten Baums gibt es genau einen Weg von der Wurzel zu v
- *Erzeugender / aufspannender Baum (Spannbaum)* eines Digraphen G: Spannwald von G mit nur 1 Wurzel

Definitionen (5)

Beobachtung: Zu jedem zusammenhängenden Graphen gibt es (mind.) einen Spannbaum:



ungerichteter Graph G_u

Markierte Graphen

Sei G = (V, E) ein (un-)gerichteter Graph, M_V und M_E Mengen und $\mu: V \rightarrow M_V$ und $g : E \rightarrow M_E$ Abbildungen.

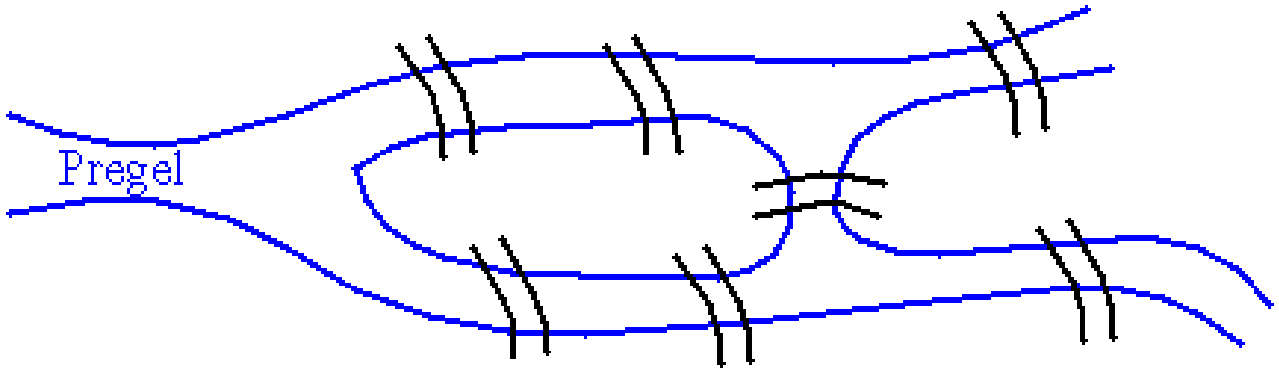
- G' = (V, E, μ) heißt knotenmarkierter Graph
- G'' = (V, E, g) heißt kantenmarkierter Graph
- G''' = (V, E, μ, g) heißt knoten- und kantenmarkierter Graph

M_V und M_E sind die Markierungsmengen (z.B. Alphabete oder Zahlen)

Algorithmische Probleme für Graphen I

Gegeben sei ein (un)gerichteter Graph G = (V, E)

- Man entscheide, ob G zusammenhängend ist
- Man entscheide, ob G azyklisch ist
- Man finde zu zwei Knoten, v, w $\in$ V einen *kürzesten Weg* von v nach w (bzw. "günstigster" Weg bzgl. Kantenmarkierung)
- Man entscheide, ob G einen *Hamiltonschen Zyklus* besitzt, d.h. einen Zyklus der Länge |V|
- Man entscheide, ob G einen *Eulerschen Weg* besitzt, d.h. einen Weg, in dem jede Kante genau einmal verwendet wird, und dessen Anfangs- und Endpunkte gleich sind (*Königsberger Brückenproblem*)



Algorithmische Probleme für Graphen II

- *Färbungsproblem*: Man entscheide zu einer vorgegebenen natürlichen Zahl k ("Anzahl der Farben"), ob es eine Knotenmarkierung $\mu: V \rightarrow \{1, 2, ..., k\}$ so gibt, dass für alle $(v, w) \in E$ gilt: $\mu(v) \neq \mu(w)$
- *Cliquenproblem*: Man entscheide für ungerichteten Graphen G zu vorgegebener natürlichen Zahl k, ob es einen Teilgraphen G' ("k-Clique") von G gibt, dessen Knoten alle paarweise durch Kanten verbunden sind
- *Matching-Problem*: Sei G = (V, E) ein Graph. Eine Teilmenge $M \subseteq E$ der Kanten heißt Matching, wenn jeder Knoten von V zu höchstens einer Kante aus M gehört. Problem: finde ein maximales Matching
- *Traveling Salesman Problem*: Bestimme optimale Rundreise durch n Städte, bei der jede Stadt nur einmal besucht wird und minimale Kosten entstehen

Hierunter sind bekannte NP-vollständige Probleme, z.B. das Cliquenproblem, das Färbungsproblem, die Hamilton-Eigenschaftsprüfung und das Traveling Salesman Problem

Speicherung von Graphen I

Knoten- und Kantenlisten

- Speicherung von Graphen als Liste von Zahlen (z.B. in Array oder verketteter Liste)
- Knoten werden von 1 bis n durchnummeriert; Kanten als Paare von Knoten

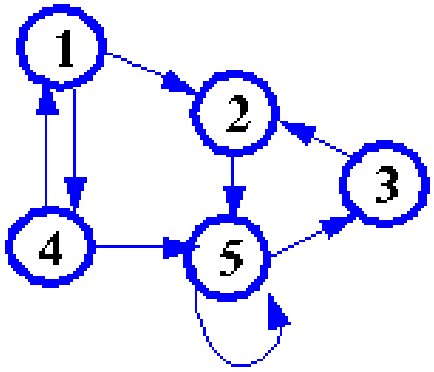
Kantenliste

- Liste: Knotenzahl, Kantenzahl, Liste von Kanten (je als 2 Zahlen)
- Speicherbedarf: 2 + 2m (m = Anzahl Kanten)

Knotenliste

- Liste: Knotenzahl, Kantenzahl, Liste von Knoteninformationen
- Knoteninformation: Ausgangsgrad und Zielknoten $ag(i)$, v_1 .. $v_{ag(i)}$
- Speicherbedarf: 2 + n+m (n= Anzahl Knoten, m = Anzahl Kanten)

Beispiel



Kantenliste:

Knotenliste:

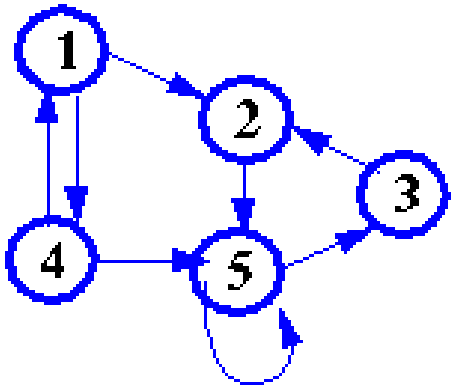
Speicherung von Graphen II

Adjazenzmatrix

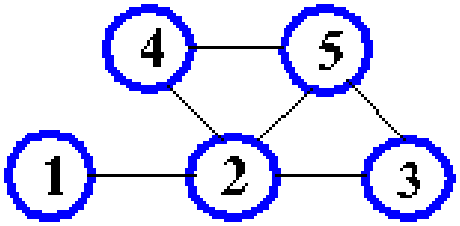
Ein Graph G = (V,E) mit |V| = n wird in einer Boole'schen n x n -Matrix AG =(a_{ij}), mit $1 \leq i,j \leq n$

gespeichert, wobei $a_{ij} = \begin{cases} 1 \text{ falls } (i,j) \in E \\ 0 \text{ falls } (i,j) \notin E \end{cases}$

Beispiel



A_G	1	2	3	4	5
1	0	1	0	1	0
2	0	0	0	0	1
3	0	1	0	0	0
4	1	0	0	0	1
5	0	0	1	0	1



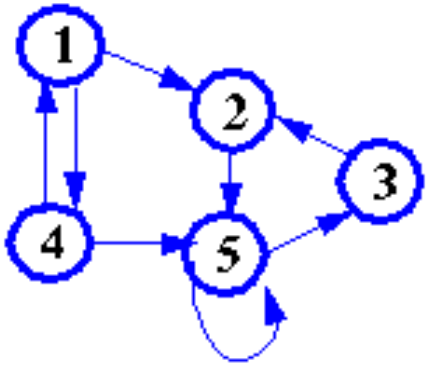
Speicherplatzbedarf $O(n^2)$

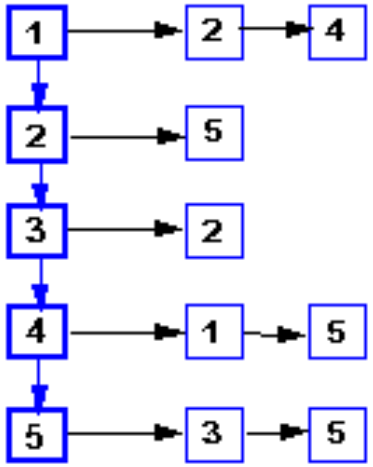
- jedoch nur 1 Bit pro Position (statt Knoten/Kantennummern)
- unabhängig von Kantenmenge
- für ungerichtete Graphen ergibt sich symmetrische Belegung (Halbierung des Speicherbedarfs möglich)

Speicherung von Graphen III

Adjazenzlisten

- verkettete Liste der n Knoten (oder Array-Realisierung)
- pro Knoten: verkettete Liste der Nachfolger (repräsentiert die von dem Knoten ausgehenden Kanten)
- Speicherbedarf: n+m Listenelemente





Speicherung von Graphen: Vergleich

Komplexitätsvergleich

Operation	Kantenliste	Knotenliste	Adjazenzmatrix	Adjazenzliste
Einfügen Kante	O(1)	O(n+m)	O(1)	O (1) / O (n)
Löschen Kante	O(m)	O(n+m)	O(1)	O(n)
Einfügen Knoten	O(1)	O(1)	$O(n^2)$	O(1)
Löschen Knoten	O(m)	O(n+m)	$O(n^2)$	O(n+m)
Speicherplatzbedarf	**O(m)**	**O(n+m)**	$O(n^2)$	**O(n+m)**

- Löschen eines Knotens löscht auch zugehörige Kanten
- Änderungsaufwand abhängig von Realisierung der Adjazenzmatrix und Adjazenzliste

Welche Repräsentation geeigneter ist, hängt auch vom Problem ab:
- Frage: Gibt es Kante von a nach b: Matrix
- Durchsuchen von Knoten in durch Nachbarschaft gegebener Reihenfolge: Listen

Transformation zwischen Implementierungsalternativen möglich